

*OMG CORBAmed DTF*

## *Healthcare Resource Access Control (HRAC)*

*Initial Submission*

*2AB*

*Baptist Health Systems of South Florida*

*CareFlow/Net, In.*

*IBM*

*OMG TC Document corbamed/98-xx-xx*

*18 October 1998*

© Copyright 1998 by 2AB, Inc.

© Copyright 1998 by Baptist Health Systems of South Florida

© Copyright 1998 by Care Data Systems, Inc.

© Copyright 1998 by IBM

The submitting companies listed above have all contributed to this "initial" submission. These companies recognize that this initial submission is the joint intellectual property of all the submitters, and may be used by any of them in the future, regardless of whether they ultimately participate in a final joint submission.

The companies listed above hereby grant a royalty-free license to the Object Management Group, Inc. (OMG) for worldwide distribution of this document or any derivative works thereof, so long as the OMG reproduces the copyright notices and the below paragraphs on all distributed copies.

The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. The companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

This document contains information which is protected by copyright. All Rights Reserved. Except as otherwise provided herein, no part of this work may be reproduced or used in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems without the permission of one of the copyright owners. All copies of this document must include the copyright and other information contained on this page.

The copyright owners grant member companies of the OMG permission to make a limited number of copies of this document (up to fifty copies) for their internal use as part of the OMG evaluation process.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013.

CORBA, OMG and Object Request Broker are trademarks of Object Management Group.

# History

<b>Initial Submission Draft 1</b>	<b>18 October 1998</b>
<p>The Editor of this document is Carol Burt of 2AB.</p> <p>The initial submission to the OMG Healthcare Resource Access Control Facility (HRAC) is the result of collaboration between the four submitting companies listed on the cover and the list of supporting companies named in the initial submission.</p>	

# Table of Contents

<b>1. Preface</b>	<b>5</b>
1.1 Submission Contact Points	5
1.2 Supporting Organizations (Dave Chizmadia)	5
1.3 Conventions	5
1.4 Terminology (Dave Chizmadia)	6
1.5 Proof of Concept (Bob Blakley – current text by Carol Burt)	6
1.6 Changes to Adopted OMG Specifications (Carol Burt)	6
1.7 Response to RFP Requirements (Bob Blakley – current text by Carol Burt)	6
<b>2. Overview of Response</b>	<b>9</b>
2.1 Introduction (Konstantin Beznosov)	9
2.2 Problems Being Addressed(Konstantin Beznosov)	9
2.3 Problems Not Being Addressed(Konstantin Beznosov)	9
2.4 Design Goals (Bob Blakley)	9
2.5 Reference Model (Dave Chizmadia)	9
2.6 Discussion of RFP Scope (Carol Burt)	9
2.7 HRAC Frequently Asked Questions (FAQ)	9
2.8 General Usage Discussion (Bob Blakley)	10
2.9 Healthcare Specific Usage Scenarios (Konstantin Beznosov, John Barkley, Juggy Jaganathan)	10
2.10 HRAC Conceptual Model (Carol Burt)	10
<b>3. DfResourceAccessControl module (C Burt)</b>	<b>11</b>
3.1 Types	12
3.2 AccessDecision interface	16
3.3 DecisionRulesAccess interface	17
3.4 DecisionRulesAdmin interface	18
3.5 DynamicEvaluator interface	18
<b>4. DfHealthCareResource module (??)</b>	<b>20</b>
<b>5. Conformance Classes (Bob Blakley)</b>	<b>21</b>
<b>6. Appendix - Use Case Examples (Konstantin Beznosov, John Barkley, Juggy Jaganathan)</b>	<b>22</b>
<b>7. Appendix - Complete IDL (Carol Burt)</b>	<b>23</b>

# 1. Preface

---

This submission is a response to CORBAmed RFP1, Patient Identification Service (PIDS), Object Management Group (OMG) document number corbamed/96-11-08.

## 1.1 Submission Contact Points

Carol Burt 2AB 3178-C Highway 31 South Pelham, AL 35124 205 621 7455 cburt@2ab.com	Konstantin Beznosov Baptist Health Systems of South Florida  Miami, FL
V. "Juggy" Jagannathan CareFlow Net, Inc. 235 High Street, Suite 225 Morgantown, WV 26505 304 293 7535 juggy@careflow.com	Bob Blakley IBM  Austin, TX

## 1.2 Supporting Organizations (Dave Chizmadia)

The following organizations have been involved in the process of developing, prototyping and/or reviewing this submission. The submitters of this response thank them for participating and giving their valuable input. A special thank you goes out to those organizations.

- Concept Five
- Inprise
- National Security Agency
- Philips Medical Systems
- ???

## 1.3 Conventions

IDL appears using this font and in a border.
--

## 1.4 Terminology (Dave Chizmadia)

**Access Decision Object (ADO) -**

**Secured Resource -**

**Client -** Any system or application that accesses or requests service from a Person Identification Service.

**Component -** A cohesive set of software services

**Naming Authority -** Any organization that assigns names determines the scope of uniqueness of the names and takes the responsibility for making sure the names are unique within its name space. In the same way that ID values are meaningful only within the context of their ID Domains, names are unique only within the context of their *naming authority*.

**System -** An application or set of applications that interact with each other, interact with the HRAC or implement HRAC. *System* in this context is synonymous with *application*. Examples of systems might include a hospital or clinical information system, an ancillary system such as a lab or radiology system, or a financial/administrative system such as an ADT.

## 1.5 Proof of Concept (Bob Blakley – current text by Carol Burt)

The initial submission is based on experience gained in implementation of proprietary access control systems by 2AB, Concept Five, and IBM and with requirements input from end user organizations such as Baptist Health Systems of South Florida, and Healthcare Vendors such as Phillips Medical Systems and CareFlow|Net. The interfaces in the initial submission will be prototypes by at least one submitting organization prior to the final submission.

## 1.6 Changes to Adopted OMG Specifications (Carol Burt)

No changes to the existing OMG specifications are needed by this specification.

## 1.7 Response to RFP Requirements (Bob Blakley – current text by Carol Burt)

### MANDATORY RFP REQUIREMENTS

*Use of the CORBA Security service credentials as the source for identifying caregivers' privileges*

The AccessDecision Interface takes a Security::AttributeList as the source for identifying caregivers' privileges. This attribute list is directly accessible from the Security::Credentials. The Credentials is not passed because the AccessDecision object is not locality constrained and to pass the object reference of a locality constrained object would violate CORBA Security. It would, however, be possible for a locality constrained SecurityLevel2::AccessDecision object to extract the Security::AttributeList from the Security::Credentials and use the HRAC AccessDecision object as part of an access control implementation.

*Ability to define secured resource categories.*

The ResourceName is a sequence of string where the first string in the sequence is required to be a NamingAuthority::QualifiedNameStr. This allows an implementation to provide groupings of

secured resources and for a client to determine from the resource name how those groupings are arranged into hierarchies. The use of the NamingAuthority module allows these groupings to be unique.

*An interface for defining access control rules for secured resources based on credentials.*

The DecisionRuleAdmin interface provides this capability. Rules are based on the rights associated with the SecAttributes that are part of the credentials. Rules are expressed as sequences of rights which may be required to exist in combinations. The effective rights of the SecAttribute may be a one-to-one mapping from the attribute or may be obtained from a SecurityAdmin::AccessPolicy get\_effective\_rights method of CORBASec.

*A set of Healthcare specific secured resources.*

The initial submission does not currently provide this set. The final submission will contain a set of standard set of ResourceName constants.

*An interface to an access control decision facility that may be used to request access control decisions.*

The AccessDecision interface provide this capability

### OPTIONAL RFP REQUIREMENTS

*Provide the ability for secured resources to be grouped for the purpose of defining access control rules*

This submission supports the concept of grouping in the ResourceName. It does not mandate the way an implementation interprets these grouping or the relationship between secured resources and the decision rules that protect them.

*An interface for defining access control rules based on attributes of the Principle (in addition)*

The specification defines how decision rules can be constructed which achieve this within the required rights model of CORBASec. A one-to-one mapping from a SecAttribute to a Right would be defined. The initial submission does not specify how an implementation would make this mapping, but it is under consideration whether this would be a good idea for a revised submission to standardize this mapping for interoperability purposes.

*An interface that extends the definition of access control rules to include context sensitive access control based on a) the day and time when the resource is accessed, b) the location of an invoking principal, c) the values of request parameters.*

The decision rules interface will allow time based rules and the DecisionRulesAccess is based on effective rules so that those not in effect may be masked. The specification also allows dynamic rights that are evaluated by a DynamicEvaluator object provided by the implementation, the application, and/or the Enterprise in which HRAC is deployed. Thus the submitters feel they have provided the architectural foundation for very sophisticated context sensitive access control external from the application.

*An interface that extends the definition of the access control rules to include notion of the relationship between a patient and a caregiver*

The DynamicEvaluator interface was designed specifically as a generic way to support relationship based (and other dynamic rights based) components of decision rules. When a right is dynamic (such as the relationship between a patient and caregiver), it is expressed as a dynamic Right which is used in defining the decision rules. A DynamicEvaluator is provided for this Right and at the time of access decision by the ADO, the evaluator is consulted to determine if the right is currently valid. See discussion of DecisionRulesAdmin set\_dynrights\_support() and DynamicEvaluator has\_right().

*A reference object model for the healthcare domain that provides a sufficient foundation for access decision logic.*

The object model is not formally expressed in UML in this initial submission as it would require work to also formally express the CORBASec model upon which this submission depends. As in CORBASec, the object model is expressed in diagrams and text.

*An interface that permits management of policy, which controls how multiple access control policy decisions governing access to the same resource are reconciled.*

The submitters felt that such expression of policy was outside the scope of the submission.

### **DISCUSSION POINTS REQUESTED**

*How new CORBAMED specifications will employ the submitted specification .*

To be written

*How existing CORBAMED specifications are to be modified.*

The submitters do not believe any modification is necessary for existing CORBAMED specifications to use the services of an HRAC, however it might be useful for some standard ResourceNames to be defined within the CORBAMED community for common resources within standard services. Such definition would be a compatible extension of existing specifications.

*Scalability and Performance of the proposal*

To be written

*Mechanisms provided for extensibility*

To be written

## 2. Overview of Response

---

### 2.1 *Introduction (Konstantin Beznosov)*

To be written

### 2.2 *Problems Being Addressed(Konstantin Beznosov)*

To be written.

### 2.3 *Problems Not Being Addressed(Konstantin Beznosov)*

To be written

The following problems are specifically **NOT** addressed by this response:

### 2.4 *Design Goals (Bob Blakley)*

To be written (see e-mail list for starter goals)

### 2.5 *Reference Model (Dave Chizmadia)*

(pretty picture goes here)

*Reference Model for HRAC.*

Nice works explaining model go here.

### 2.6 *Discussion of RFP Scope (Carol Burt)*

To be written

### 2.7 *HRAC Frequently Asked Questions (FAQ)*

From e-mail list

## *2.8 General Usage Discussion (Bob Blakley)*

To be written

## *2.9 Healthcare Specific Usage Scenarios (Konstantin Beznosov, John Barkley, Juggy Jaganathan)*

To be written

## *2.10 HRAC Conceptual Model (Carol Burt)*

To be written

### 3. DfResourceAccessControl module (C Burt)

```
//File: DfResoureAccessControl

#ifndef _DF_RESOURCE_ACCESS_CONTROL_IDL_
#define _DF_RESOURCE_ACCESS_CONTROL_IDL_

#include <orb.idl>
#include <Security.idl>

#pragma prefix "omg.org"

module DfResourceAccessControl
{

interface AccessDecision {
...
};

interface DecisionRulesAccess {
...
};

interface DecisionRulesAdmin : DecisionRulesAccess {
...
};

interface DynamicEvaluator {
...
};

};

#endif // _DF_RESOURCE_ACCESS_CONTROL_IDL_
```

The DfResourceAccessControl contains four interfaces defined below and has type dependencies on the CORBA Security Service and the CORBAMED NamingAuthority modules.

```
#include <Security.idl>
```

The types declared within the Security service and used by the HRAC include

```
Security::AttributeList
Security::Right
Security::RightsList
Security::RightsCombinator
```

These types are used for consistency with CORBASec and have the same meaning when used in HRAC interfaces. They are typedef'd in this specification for ease of use.

```
#pragma prefix "omg.org"
```

In order to prevent name pollution and name clashing of IDL types this module (and all modules defined in this specification) uses the pragma prefix that is the omg DNS name.

## 3.1 Types

There are a number of structured types used widely through out the DfResourceAccessControl Model. These types are described in this section:

### 3.1.1 Basic Types & Types used from the CORBA Security service

```
/**
 * *****
 * //          Basic Types & Types used from Security
 * *****
 */

typedef sequence<boolean> Booleans;

typedef TimeBase::IntervalT IntervalT;
typedef sequence<IntervalT> TimeIntervals;

typedef Security::AttributeList AttributeList;
typedef Security::Right Right;
typedef Security::RightsList RightsList;
typedef Security::RightsCombinator RightsCombinator;
```

#### **Booleans**

A sequence of boolean used as a return value when multiple decisions are requested. This type is used as a return value in AccessDecision and DynamicEvaluator interface methods.

#### **IntervalT**

TBW

#### **TimeIntervals**

TBW

#### **AttributeList**

The Security::AttributeList is defined as follows in CORBA Security 1.2 (ptc/98-01-02).

```
typedef sequence<octet> Opaque;
typedef sequence<AttributeType> AttributeTypeList;

struct SecAttribute {
    AttributeType    attribute_type;
    Opaque           defining_authority;
    Opaque           value;
    // the value of this attribute can be
    // interpreted only with knowledge of type
};

typedef sequence <SecAttribute> AttributeList;
```

#### **Right**

The Security::Right is defined as follows in CORBA Security 1.2 (ptc/98-01-02). An explanation of Rights and Rights Families is in section 15.6.4 on pages 15-123 and 15-124 of the CORBA Security 1.2 specification. The SecurityAdmin::AccessPolicy interface is described in section 15.6.4.3 on page 15-126 of that specification. A detailed description of how access decisions are made using a required rights model in section 15.6.4.5 starting on page 15-127.

```

struct ExtensibleFamily {
    unsigned short    family_definer;
    unsigned short    family;
};
struct Right {
    ExtensibleFamily    rights_family;
    string              right;
};

```

**RightsList**

```
typedef sequence <Right> RightsList;
```

**RightsCombinator**

```

enum RightsCombinator {
    SecAllRights,
    SecAnyRight
};

```

A RightsCombinator when used in combination with a RightsList specifies how the elements of the list are related for the purpose of access decision. An OperationRuleComponent in this specification uses a combinator and rights list to specify one component of a decision rule on an operation of a secured resource.

3.1.2 Types that identify and manage information about secured resources

```

//*****
//    Type that identifies a Secured Resource
//*****
typedef sequence<string> ResourceName;

```

**ResourceName**

A ResourceName is used to identify a **secured resource**. ResourceName is a sequence of string allowing for groupings of resources. It is required that the first string in the sequence is formatted as a NamingAuthority::QualifiedNameStr. This ensures globally unique resource names (and/or groups) See NamingAuthority module in corbamed/98-02-29. Note: We should discuss whether we want to make this a stronger requirement by putting the type in the IDL.

3.1.3 Types related to Resource Access Decision Rules

```

//*****
//    Types related to Resource Access Decision Rules
//*****

```

```

struct OperationRuleComponent {
    RightsCombinator    combinator;
    RightsList          rights;
};
typedef sequence<OperationRuleComponent> OperationRule;

enum    PermissionControlModel { GRANT, DENY };

struct Rule {
    PermissionControlModel control_model;
    OperationRule          the_rules;
};

struct ResourceRuleComponent {
    string                operation;
    OperationRule         the_rule;
};
typedef sequence<ResourceRuleComponent> ResourceRules;

struct Rules {
    PermissionControlModel control_model;
    ResourceRules          the_rules;
};

```

**OperationRuleComponent**

An operation rule component specifies a single component of an operation **access decision rule**. For example, the component may be as follows:

Combinator	RightsList
All	accessid:carol, role:architect, role:executive

**OperationRule**

An operation rule is a sequence of OperationRuleComponent. There is an implied “OR” between the members of the sequence. For example, if a rule had a sequence of two OperationRuleComponents as follows:

Combinator	RightsList
All	accessid:carol, role:architect, role:executive

Combinator	RightsList
Any	accessid:bob, role:goodguy

The OperationRule would be interpreted as ((accessid:carol & role:architect & role:executive) | accessid:bob | role:goodguy)

**PermissionControlModel**

Resource Decision Rules have a grant/deny PermissionControlModel that is expressed as part of the definition of the rules. GRANT means: the decision rules for this resource are used to GRANT access. DENY means: the decision rules for this resource are used to DENY access. So, the control is expressed on the resource decision rule and tells the service how to evaluate the rule against effective rights. The way that a decision is made is that the effective\_rights of the

principle is compared against the rule. If the PermissionControlModel is GRANT, then the access is granted (access\_allowed = TRUE) if the rights evaluated against the rules are true. If the PermissionControlMode is DENY, then access is denied (access\_allowed = FALSE) if the rights evaluated against the rules are true.

The “decision rules” are specified by defining the rights required to access a secured resource without regard to how rights are assigned to a SecAttribute.

#### **Rule**

A Rule contains the PermissionControlModel for the operation and the OperationRule for a single operation. This is the type returned from the get\_rule method where the client is asking for a single rule for a Resource/Operation pair. The PermissionControlModel is at the Resource level, but is repeated in a Rule so that an Access Decision Object could retrieve if desired the rule for a single operation of a resource without retrieving all rules for the resource or making a separate method invocation to get the permission control model.

#### **ResourceRuleComponent**

A ResourceRuleComponent specifies a single component of the access decision rules for a secured resource. It includes the operation and the OperationRule. This type is only used to build other types and is not used directly in any IDL interface.

#### **ResourceRules**

ResourceRule is a sequence of all the ResourceRule components for a secured resource

#### **Rules**

Rules contain the PermissionControlModel for the resource and the ResourceRules.

### 3.1.4 Types used to request an Access Decision

```

//*****
//      Types used to request an Access Decision
//*****

struct AccessDef {
    ResourceName  resource_name;
    String        operation;
};
typedef sequence<AccessDef> MultipleAccessDef;
```

#### **AccessDef**

The AccessDef struct is provided to allow multiple access definitions to be defined. It contains the ResourceName and the operation name for the secured resource access being requested.

#### **MultipleAccessDef**

MultipleAccessDef is the type used to request multiple access decisions in a single operation. See the multiple\_access\_allowed() method of the AccessDecision interface.

### 3.1.5 Types related to Dynamic Evaluation of Rights

```
/** *****  
// Types related to Dynamic Evaluation of rights  
/** *****  
  
typedef sequence<octet> ResourceKey;  
  
struct DynamicInfo {  
    DynamicEvaluator    evaluator;  
    ResourceKey         key;  
    RightsList          dyn_rights;  
};
```

#### **ResourceKey**

The ResourceKey is registered as a resource identifier when a dynamic evaluator is used. See the usage in the DecisionRulesAdmin set\_dynrights\_support() and the DynamicEvaluator evaluate() methods. This is the only usage of this type – it is merely saved by HRAC and provided as a parameter with the invocation of a method on a dynamic evaluator. This is a convenience to the application who may have an internal identifier of the secured resource that is more convenient to use than a ResourceName at the time the dynamic evaluator is called.

#### **DynamicInfo**

This structure contains all of the information necessary to register a dynamic evaluator. It contains object reference of the DynamicEvaluator, the ResourceKey of the resource, and a list of dynamic rights which it can be evaluated by the evaluator for the resource. See the set\_dynrights\_support() method of the DecisionRulesAdmin interface.

### 3.1.6 Exceptions

The following exceptions are generally useful by most or all of the interfaces of this module.

```
TBD
```

## 3.2 AccessDecision interface

```
/** *****  
// interface AccessDecision  
/** *****  
  
interface AccessDecision {  
  
    boolean access_allowed(  
        in ResourceName  rname,  
        in string         operation,  
        in AttributeList attributes  
    );  
};
```

```
Booleans multiple_access_allowed(  
    in MultipleAccessDef requested_access,  
    in AttributeList     attributes  
);  
  
};
```

The Access Decision object is used to request decisions on access based on a ResourceName, an Operation, and a list of SecAttributes. It is out of the scope of this submission to mandate how the ADO determines the effective\_rights of the principal based on the SecAttributes.

**access\_allowed()**

A single access decision is requested and a boolean is returned

**Multiple\_access\_allowed()**

Multiple access decisions are requested in a single method invocation and a sequence of booleans are returned. The boolean sequence maps one to one in the same order to the provided sequence of ResourceName/operation pairs.

### 3.3 *DecisionRulesAccess interface*

```
/**  
 * interface DecisionRulesAccess  
 */  
interface DecisionRulesAccess {  
  
    DynamicInfo get_dynrights_support();  
  
    Rule get_effective_rule(  
        in ResourceName rname,  
        in string operation  
    );  
  
    Rules get_effective_rules(  
        in ResourceName rname  
    );  
  
};
```

The DecisionRulesAccess allows an Access Decision Object to retrieve the information necessary from the rules base to make an access decision.

**get\_dynrights\_support()**

This provides to the client an object reference of Hmmmm Note: There is a missing input parameter here. I need to think about this.

**get\_effective\_rule()**

The Rule for a single operation on a resource is provided with this method

**get\_effective\_rules()**

All of the rules for a secured resource are provided with this method.

### 3.4 *DecisionRulesAdmin* interface

```
/** *****  
//      interface DecisionRulesAdmin : DecisionRulesAccess  
/** *****  
interface DecisionRulesAdmin : DecisionRulesAccess {  
  
    void set_resource_key (  
        in ResourceName rname,  
        in ResourceKey key  
    );  
  
    void set_dynrights_support (  
        in DynamicInfo dynamic_info  
    );  
  
    void set_rule (  
        in PermissionControlModel control_model,  
        in ResourceName      rname,  
        in string             operation,  
        in TimeIntervals     effective_time,  
        in Rule               the_rule  
    );  
  
};
```

The DecisionRulesAdmin interface extends the DecisionRulesAccess interface ...

#### **set\_resource\_key()**

This associates a ResourceKey with a ResourceName. This key will be used in parameters of the DynamicEvaluator instead of the ResourceName.

#### **set\_dynrights\_support()**

This sets the information necessary for dynamic evaluation of rights. Note: Missing a parameter.

#### **set\_rule()**

This allows the definition of decision rules. Note: Time needs to be expanded for the rule definition.

### 3.5 *DynamicEvaluator* interface

```
/** *****  
//      interface DynamicEvaluator  
/** *****  
interface DynamicEvaluator {  
  
    boolean evaluate(  
        in ResourceKey      resource_key,  
        in RightsList       effective_rights,  
        in Right            dynamic_right);  
  
    Booleans multiple_evaluate(  
        in ResourceKey      resource_key,  
        in RightsList       effective_rights,  
        in RightsList       dynamic_right);  
  
};
```



The DynamicEvaluator interface. Note: Need an example.

**evaluate()**

The evaluator determines if, the dynamic\_right is currently in force based on the resource identified and the effective rights passed to the evaluator

**Multiple\_evaluate()**

This is a request for multiple evaluations with a single method call.

## 4. *DfHealthCareResource* module (??)

---

This should contain the healthcare specific ResourceNames that are standardized in this specification.

## 5. Conformance Classes (Bob Blakley)

---

6. Appendix - Use Case Examples (Konstantin Beznosov, John Barkley, Juggy Jaganathan)

## 7. Appendix - Complete IDL (Carol Burt)

---

```
//File: DfResourceAccessControl.idl
//
#ifdef _DF_RESOURCE_ACCESS_CONTROL_IDL_
#define _DF_RESOURCE_ACCESS_CONTROL_IDL_

#include <orb.idl>

#include "Security.idl"

#pragma prefix "omg.org"

module DfResourceAccessControl {

//*****
//* Forward interfaces
//*****
interface DynamicEvaluator;

//*****
// Basic Types & Types used from Security
//*****

typedef sequence<boolean> Booleans;

typedef TimeBase::IntervalT IntervalT;
typedef sequence<IntervalT> TimeIntervals;

typedef Security::AttributeList AttributeList;
typedef Security::Right Right;
typedef Security::RightsList RightsList;
typedef Security::RightsCombinator RightsCombinator;

//*****
// Types that identify and manage information about secured resources
//*****

typedef sequence<string> ResourceName;

//*****
// Types related to Resource Access Decision Rules
//*****

struct OperationRuleComponent {
    RightsCombinator combinator;
    RightsList rights;
};
typedef sequence<OperationRuleComponent> OperationRule;

enum PermissionControlModel { GRANT, DENY };

struct Rule {
    PermissionControlModel control_model;
    OperationRule the_rules;
};

struct ResourceRuleComponent {
    string operation;
    OperationRule the_rule;
};
typedef sequence<ResourceRuleComponent> ResourceRules;

struct Rules {
```

```

        PermissionControlModel control_model;
        ResourceRules          the_rules;
};

//*****
//      Types used to request an Access Decision
//*****

struct AccessDef {
    ResourceName  resource_name;
    string        operation;
};
typedef sequence<AccessDef> MultipleAccessDef;

//*****
//      Types related to Dynamic Evaluation of rights
//*****

typedef sequence<octet> ResourceKey;

struct DynamicInfo {
    DynamicEvaluator  evaluator;
    ResourceKey       key;
    RightsList        dyn_rights;
};

//*****
//      interface AccessDecision
//*****

interface AccessDecision {

    boolean access_allowed(
        in ResourceName  rname,
        in string        operation,
        in AttributeList attributes
    );

    Booleans multiple_access_allowed(
        in MultipleAccessDef requested_access,
        in AttributeList     attributes
    );

};

//*****
//      interface DecisionRulesAccess
//*****

interface DecisionRulesAccess {

    DynamicInfo get_dynrights_support();

    Rule  get_effective_rule(
        in ResourceName rname,
        in string       operation
    );

    Rules get_effective_rules(
        in ResourceName rname
    );

};

//*****
//      interface DecisionRulesAdmin : DecisionRulesAccess

```

```

//*****
interface DecisionRulesAdmin : DecisionRulesAccess {

    void set_resource_key (
        in ResourceName rname,
        in ResourceKey key
    );

    void set_dynrights_support (
        in DynamicInfo dynamic_info
    );

    void set_rule (
        in PermissionControlModel control_model,
        in ResourceName rname,
        in string operation,
        in TimeIntervals effective_time,
        in Rule the_rule
    );
};

//*****
//      interface DynamicEval
//*****
interface DynamicEvaluator {

    boolean evaluate(
        in ResourceKey resource_key,
        in RightsList effective_rights,
        in Right dynamic_right);

    Booleans multiple_evaluate(
        in ResourceKey resource_key,
        in RightsList effective_rights,
        in RightsList dynamic_right);
};

};
#endif // DfResourceAccessControl

```